Getting More RDF Support from Relational Databases

François Goasdoué
Univ. Paris-Sud & Inria Saclay
PCRI, bât. 650, U. Paris-Sud
91405 Orsay Cedex France
fg@lri.fr

Ioana Manolescu
Univ. Paris-Sud & Inria Saclay
PCRI, bât. 650, U. Paris-Sud
91405 Orsay Cedex France
ioana.manolescu@inria.fr

Alexandra Roatiş
Univ. Paris-Sud & Inria Saclay
PCRI, bât. 650, U. Paris-Sud
91405 Orsay Cedex France
alexandra.roatis@inria.fr

ABSTRACT

We introduce the *database fragment of RDF*, which extends the popular Description Logic fragment, in particular with support for incomplete information. We then provide novel *sound and complete saturation- and reformulation-based techniques* for answering the *Basic Graph Pattern* queries of SPARQL in this fragment. Notably, we extend the state of the art on pushing RDF query processing within robust / efficient relational database management systems. Finally, we experimentally compare our query answering techniques using well-established datasets.

Categories and Subject Descriptors

H.2.4 [**Database Mgmt**]: Systems—*query processing*

General Terms

Algorithms, Experimentation, Theory

Keywords

RDF fragments, RDF query reformulation, RDF saturation

1. INTRODUCTION AND CONCEPTS

The Resource Description Framework (RDF) is a graph-based data model and the W3C standard for Semantic Web applications. An *RDF graph* is a set of *triples* of the form `s p o .` stating class and property assertions (Figure 2). *RDF Schema* (RDFS) allows enhancing RDF graphs by stating *semantic constraints* between classes and properties (Figure 2). Interestingly, RDF graphs can model incomplete information using *blank nodes* (labelled nulls), allowing to handle unknown classes, properties, and values.

RDF entailment is the mechanism through which, based on explicit triples and *entailment rules*, *implicit RDF triples* are derived. This allows defining the (finite) *saturation* of an RDF graph `G`, denoted $\mathtt{G}^\infty$, obtained by making all the implicit triples explicit. From the RDF standard perspective, an RDF graph is semantically equivalent to its saturation.

We consider a subset of the SPARQL language, consisting of (unions of) *conjunctive queries*, defined by basic graph patterns (BGPs), i.e., sets of *triple atoms*.

A conjunctive query is denoted $q(\bar{x})$:- $t_1, \ldots, t_\alpha$ ($\bar{x}$ is empty in boolean queries). The variables $\bar{x}$ in the head of the query are called *distinguished variables*, and are a subset of the variables occurring in $t_1, \ldots, t_\alpha$. Evaluation *treats the blank nodes in a query as non-distinguished variables*. The (complete) *answer set* of a query q against `G` is obtained by the evaluation of q against $\mathtt{G}^\infty$, denoted by $q(\mathtt{G}^\infty)$.

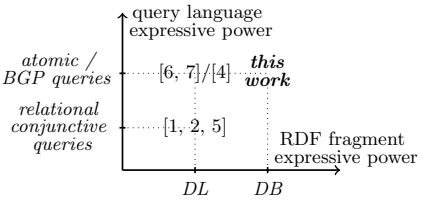

Figure 1: Positioning of our work.

Triples		Entailed triple
s rdfs:subClassOf o .,	o rdfs:subClassOf o_1 .	s rdfs:subClassOf o_1 .
s rdfs:subPropertyOf o .,	o rdfs:subPropertyOf o_1 .	s rdfs:subPropertyOf o_1 .
s rdfs:domain o .,	o rdfs:subClassOf o_1 .	s rdfs:domain o_1 .
s rdfs:range o .,	o rdfs:subClassOf o_1 .	s rdfs:range o_1 .
s rdfs:domain o .,	s_1 rdfs:subPropertyOf s .	s_1 rdfs:domain o .
s rdfs:range o .,	s_1 rdfs:subPropertyOf s .	s_1 rdfs:range o .
s_1 rdfs:subClassOf s_2 .,	s rdf:type s_1 .	s rdf:type s_2 .
p_1 rdfs:subPropertyOf p_2 .,	s p_1 o .	s p_2 o .
p rdfs:domain s .,	s_1 p o_1 .	s_1 rdf:type s .
p rdfs:range s .,	s_1 p o_1 .	o_1 rdf:type s .

Figure 3: Schema- and instance-level entailment.

In this paper we define the *database (DB) fragment of RDF*, strictly more expressive than the Description Logic fragment, that captures essential RDF features, such as incomplete information, and treats class and property names like any other value in the database. Then, we study *query answering in this DB fragment*, through the saturation- and reformulation-based approaches, for the BGP queries.

The positioning of our work w.r.t. the literature is sketched in Figure 1. More details are available at [3].

2. THE DATABASE FRAGMENT OF RDF

We define the *database (DB) fragment of RDF* by:

- *restricting RDF entailment to the entailment rules dedicated to RDF Schemas* (Figure 3);
- *not restricting RDF graphs in any way.*

An RDF graph belonging to our DB fragment is an *RDF database* $\mathtt{db} = \langle \mathtt{S}, \mathtt{D} \rangle$, where the *schema-level* `S` and *instance-level* `D` are disjoint sets of triples, respectively made of the RDFS statements and the RDF statements in `db`.

The saturation of a database `db` with this restricted rule set is denoted $\mathtt{db}^\propto$. The *evaluation of a query q against* `db` is the normative evaluation of q against the RDF graph `db`, i.e., $q(\mathtt{db})$, and the *answer set of q against* `db` is $q(\mathtt{db}^\propto)$.

3. QUERY ANSWERING IN RDF

We investigate *saturation-* and *reformulation-based* query answering against RDF databases. Each technique performs a specific *pre-processing* step, either on the database or on

RDF statements	
Class assertion	s rdf:type o .
Property assertion	s p o .

RDFS statements	
Subclass constraint	s rdfs:subClassOf o .
Subproperty constraint	s rdfs:subPropertyOf o .
Domain typing constraint	s rdfs:domain o .
Range typing constraint	s rdfs:range o .

G =

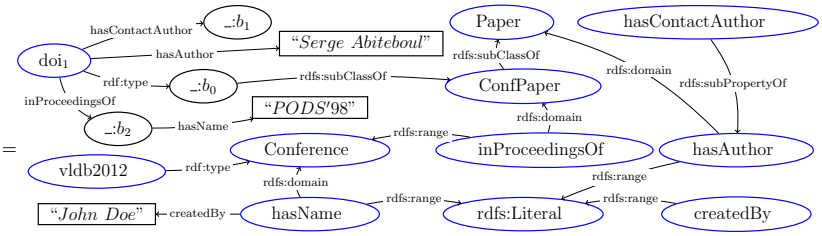

Figure 2: RDF & RDFS Statements (left). RDF graph G (right).

the queries, to deal with entailed triples; after which query answering is reduced to query evaluation. We focus on answering *instance-level queries*, i.e., the usual database queries.

Saturation-based query answering. Given a database db, Saturate(db) [3] is an algorithm that exhaustively applies 4 saturation rules on db plus all the gradually generated triples. The worst-case size (number of triples) of its output is in $O(\#\text{db}^2)$, where #db is the size of db. $\text{Saturate}_+(\text{db})$ [3] is a *multiset* variant of Saturate(db) – a triple appears as many times as it can be entailed – allowing *saturation maintenance upon updates*, to avoid recomputing the whole saturation upon every update on db.

THEOREM 1. *For a query q and a database* db,
$q(\text{db}^\infty) = q(\text{Saturate}(\text{db})) = q(\text{set}(\text{Saturate}_+(\text{db})))$ *holds.*

Reformulation-based query answering. Given a query q and a database db, Reformulate(q, db) [3] is an algorithm that reformulates q, using 13 reformulation rules, into a set of queries, such that the union of the *non-standard evaluations* (see below) of these queries on db produces $q(\text{db}^\infty)$. The worst-case size (number of queries) of the output of Reformulate(q, db) is in $O((6 * \#\text{db}^2)^{\#q})$, with #db and $\#q$ the sizes (number of triples) of db and q respectively.

Standard evaluation is based on assignments of all the query's variables and blank nodes into database values, treating blank nodes as non-distinguished variables. In contrast, when our Reformulate(q, db) algorithm brings blank nodes in queries, it refers precisely to these particular blank nodes in db. We therefore define the *non-standard evaluation* of a query q against a database db, denoted $\tilde{q}(\text{db})$, that only assigns the query variables, treating blank nodes as constants.

THEOREM 2. *For a database* db *and a query q against* db,

$$q(\text{db}^\infty) = \bigcup_{q_\sigma \in \text{Reformulate}(q,\text{db})} \tilde{q}_\sigma(\text{db}) \text{ holds.}$$

4. EXPERIMENTAL EVALUATION

We implemented our algorithms in Java 1.6 and delegate query evaluation to a PostgreSQL server v8.5. The *instance-level*, *set- and multiset-based saturation* are stored in separate tables, identically indexed with 15 indexes for efficient query evaluation opportunities. The *schema-level* is kept in memory. *All measured times are averaged over five runs.*

We present results obtained for the DBpedia [8] dataset (#S = 5666 triples, #D = 27 million triples), other results are available at [3]. Saturation added 10% to the database size in $t_{sat} = 2,742s$ and $t_{sat+} = 2,977s$, where t_{sat}, t_{sat+} is the saturation time using Saturate, resp. Saturate_+.

For *query answering*, we hand-picked 20 queries of 1 to 4 atoms. We call the *saturation threshold* of a query q, or $st(q)$, the integer n representing the minimum number of times one needs to run q for the whole saturation cost to amortize. Similarly, we define st^+/st^- for amortizing the *maintenance overhead due to one triple insertion/deletion.* Figure 4 shows that the thresholds are strongly correlated

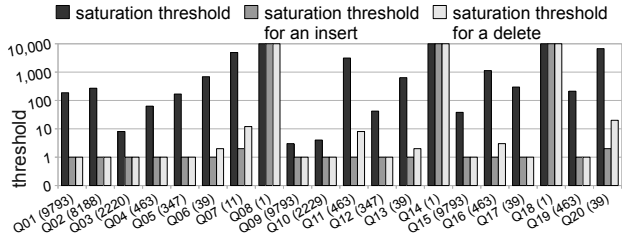

Figure 4: Saturation threshold for DBpedia queries.

with the size of the reformulated query (in parentheses on the x axis). The larger the reformulated query, the lower the threshold: saturation pays off faster when reformulation is expensive. st is always higher than st^+ and st^-, since st runs offset *the complete saturation cost*, whereas st^+ and st^- need to offset the cost of maintaining saturation for just one triple added or deleted. Finally, st^- is lower than st^+: saturation costs particularly penalize frequent deletions.

Conclusion. The saturation thresholds in RDF databases strongly depend on the size of the reformulated query (which depends on the schema), and the query selectivity. While saturation is the default choice in many RDF data management systems, for queries with small reformulations, its overhead is very high. This confirms the practical interest of our reformulation-based query answering technique.

5. CONCLUSION

Our work extends the state of the art on practical RDF data management based on RDBMS. Notably, we provide reformulation- and saturation-based query answering techniques that are robust to updates, and empirical performance thresholds between them. Further optimizations and performance comparisons are ongoing.

6. ACKNOWLEDGEMENTS

This research was partially supported by Digiteo through the DIM DW4RDF grant.

7. REFERENCES

[1] P. Adjiman, F. Goasdoué, and M.-C. Rousset. SomeRDFS in the semantic web. *JODS*, 8, 2007.

[2] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning (JAR)*, 39(3), 2007.

[3] http://www.lri.fr/~roatis/WWW2012-TR.pdf.

[4] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu. View selection in semantic web databases. *PVLDB*, 2011.

[5] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *ICDE*, 2011.

[6] Z. Kaoudi, I. Miliaraki, and M. Koubarakis. RDFS reasoning and query answering on DHTs. In *ISWC*, 2008.

[7] J. Urbani, F. van Harmelen, S. Schlobach, and H. Bal. QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases. In *ISWC*, 2011.

[8] Dbpedia 3.7. http://wiki.dbpedia.org/Downloads37.